This article is part of the IEEE Reliability Society 2009 Annual Technology Report.

The Role of Software in Recent Catastrophic Accidents

W. Eric Wong, Vidroha Debroy, and Andrew Restrepo
Department of Computer Science
University of Texas at Dallas
{ewong,vxd024000,arestrep}@utdallas.edu

1. INTRODUCTION

Recent decades bear testimony to how we have gone from merely using software, to relying on it, and ultimately becoming dependent on it, for our day to day lives. Software is also extensively used in areas such as medicine, transportation, nuclear power generation, aeronautics and astronautics, and national defense, only to name a few. Such areas are safety-critical, and extremely sensitive to errors that once may have only been man-made, but today may also be made by faulty software, or software that has been used in a faulty manner. The smallest flaw could have devastating consequences that can lead to significant damage, including the loss of life. Thus, how much trust can we put in our software; and how much damage can erroneous software cause.

We review fifteen recent catastrophic accidents, and evaluate the causative roles that software might have played in the accidents. We note that software may not be the sole cause of all accidents reported in this article, and quite often the causes are a combination of both software and human error. Nevertheless, the focus of this article is still on how faulty software, either directly or indirectly, led to the accidents. We also discuss the lessons that might be learned from the accidents, and the wisdom that we might take away from them. Our list of accidents is in no way complete. The article however aims to be as extensive and detailed as is possible. The accidents were chosen based on the scale of the damage, and on how large a role was played by software, as well as other criteria.

2. RECENT CATASTROPHIC ACCIDENTS

Shutdown of the Hartsfield-Jackson Atlanta International Airport [4]

Hartsfield–Jackson Atlanta International Airport is one of the world's busiest airports, both in terms of passengers, and number of flights. The alertness of the security screeners is tested by the random appearance of artificial bombs or other suspicious hard-to-detect devices on the X-ray machine displays, followed by a brief delay, then a message indicating that it was a test. On April 19, 2006, an employee of the Transportation Security Administration (TSA), U.S. Department of Homeland Security, identified the image of a suspicious device, but did not realize it was part of the routine testing for security screeners because the software failed to indicate such a test was underway. As a result, the airport authorities evacuated the security area for two hours while searching for the suspicious device, causing more than 120 flight delays, and forcing many travelers to wait outside the airport.

The false alarm was clearly due to a software malfunction which failed to alert the screeners by showing a warning message to indicate that the image of the suspicious device was just a test. Whenever a critical software system is using *sample* data for testing purposes, it should not create unnecessary suspicion among the end users. This sample data should be carefully chosen, and a thorough controlled testing should be conducted to ensure that an appropriate test alert message is displayed.

Loss of Communication between the FAA Air Traffic Control Center, and Airplanes [7], [13]

On Tuesday, September 14, 2004, the Los Angeles International Airport, and other airports in the region suspended operations due to a failure of the FAA radio system in Palmdale, California. Technicians on-site failed to perform the periodic maintenance check that must occur every 30 days, and the system shut down without warning as a result. The controllers lost contact with the planes when the main voice communications system shut down unexpectedly. Compounding this situation was the almost immediate crash of a backup system that was supposed to take over in such an event. The outage disrupted about 600 flights (including 150 cancellations), impacting over 30,000 passengers. Flights through the airspace controlled by the Palmdale facility were either grounded, or rerouted elsewhere. Two airplane accidents almost occurred, and countless lives were at risk.

A bug in a Microsoft system compounded by human error was ultimately responsible for the three-hour radio breakdown. A Microsoft-based replacement for an older Unix system needed to be reset approximately every 50 days to prevent data overload. A technician failed to perform the reset at the right time, and an internal clock within the system subsequently shut it down. In addition, a backup system also failed. When a system has a known problem, it is never a good idea to continue operation. Instead of relying on an improvised workaround, the bug in the software should be corrected as soon as possible to avoid a potential crisis. Learning from this experience, the FAA deployed a software patch which now addresses this issue. If the backup had been operating correctly, this accident would not have occurred. For systems where a high degree of safety is of utmost concern, solid redundancies should always be in place.

Crash of Air France Flight 447 [1]
On May 31, 2009, an Airbus A330-200 departed from the Rio de Janeiro-Galeão International Airport due to arrive at Paris 11 hours later. It crashed into the Atlantic Ocean on June 1. The aircraft was carrying 216 passengers, and 12 crew members, all of whom are presumed to be dead. In addition to the loss of the aircraft itself, Air France announced that each victim's family would be paid roughly €17,500 in initial compensation. This accident makes it the deadliest disaster for Air France, surpassing the Air France Flight 4590 in 2000 that killed 109 people.

Regarding software related contributing factors, the onboard automating reporting system transmitted several messages regarding discrepancies in the indicated air speed (IAS) readings before the aircraft disappeared. In total, 24 error messages were generated as systems failed across the aircraft. On June 4, 2009 (three days after the crash of Flight 447), Airbus issued an Accident Information Telex to operators of all Airbus models reminding pilots of the recommended Abnormal and Emergency Procedures to be taken in the case of unreliable airspeed indication. Efforts to retrieve the flight data recorders, critical to determining the exact cause of the crash, will resume in February 2010, but the chance of recovery is low. A final report on the accident is expected to be issued by the end of 2010.

Crash of Korean Air Flight 801 [10], [15], [18]
On August 5, 1997, Korean Air Flight 801 departed from Kimpo International Airport in Seoul, South Korea, bound for Guam. On board were 254 people: 2 pilots, 1 flight engineer, 14 flight attendants, and 237 passengers. The flight experienced some turbulence, but was uneventful until shortly after 1:00 am on August 6, as the jet prepared to land. At 1:42 am, the aircraft crashed into Nimitz Hill, about 3 miles (5 kilometers) short of the runway, at an altitude of 660 feet (201 meters). In terms of the loss incurred, of the 254 people on board, 228 were killed, and 23 passengers and 3 flight attendants survived the accident with serious injuries. The airplane was destroyed by impact forces, and a post-crash fire. In 2000, a lawsuit was settled in the amount of $70,000,000 US dollars on behalf of 54 families.

The National Transportation Safety Board (NTSB) determined that the probable cause was the captain’s failure to adequately brief and execute a non-precision localizer-only approach, and the first officer’s and flight engineer’s failure to effectively monitor and cross-check the captain’s execution of the approach. However, two of the identified software related causes were the FAA’s intentional inhibition of the Minimum Safe Altitude Warning (MSAW) system, and the agency’s failure to adequately manage the system. The MSAW system was developed by the FAA in response to the NTSB Safety Recommendation A-73-46. However, there was a 54-nautical mile (where one nautical mile is 1.852 kilometers or about 1.15 miles) inhibit zone at the Guam International Airport to alleviate nuisance warnings. By doing so, the usefulness of the system was limited. Simulations by the NTSB, and the FAA indicated that, without the inhibition, MSAW would have generated an alert 64 seconds before impact. This would have been sufficient for the controller to advise Flight 801, and give the crew an opportunity to take appropriate actions to avoid the crash. This accident teaches us that when changing any part of a safety-critical software system, we must examine how that change will affect the safety of the overall system.

Crash of American Airlines (AA) Flight 965 [14]
On December 20, 1995, AA965 departed from Miami International Airport bound for Cali, Colombia. At 9:40 PM, just five minutes before its scheduled arrival, the plane went down in the Andes, crashing into the west slope of a mountain. The crash led to a total of 159 deaths, and completely destroyed the airplane (a Boeing 757). Only four of the passengers and a dog survived the crash. The crash was the first U.S.-owned 757 accident, and resulted in the highest death toll of any accident in Colombia. The death toll was also the highest of any accident involving a Boeing 757 at the time, and was the deadliest air disaster involving a U.S. carrier since the downing of Pan Am Flight 103 on December 21, 1988.

The circumstances leading to the crash involved the pilots programming the navigational computer with the location of Rozo (a waypoint on their approach). Although this waypoint was designated “R” on their charts, the same identifier had also been used for a second radio beacon (Romeo) near Bogotá. When a crewmember of Flight 965 changed the approach course to the airport, he input only “R” instead of “Rozo” in the computer. The flight management system (FMS) interpreted this as a request to set a course to the “Romeo” waypoint, which has a higher frequency than the one at Rozo. When two waypoints with the same initial are nearby, the FMS automatically chooses the one with the higher frequency. The pilot selected the first “R” on the list, unwittingly setting a course for Bogotá, which caused the autopilot to execute a wide semicircular turn to the east. By the time the error was discovered, the airplane was already on a collision course with a 3,000 meter mountain.

Even though the cause of the accident was portrayed as pilot error, a large part of the blame also lies with the poor design of the software system (i.e. the FMS). If a user were to enter just an initial as opposed to the full destination name, the flight is directed to the destination which has the highest frequency among all destinations which start with that initial. Although such a feature may be useful, it does not come without risk as we learnt from this accident. The crash of flight 965 also teaches us that one cannot assume that a computer automatically ensures safe operation; and it is also important for human users to maintain adequate situational awareness.

Mishap Involving NASA’s Demonstration of Autonomous Rendezvous Technology [17]
On April 15, 2005, the Demonstration of Autonomous Rendezvous Technology (DART) spacecraft was successfully launched. DART was designed to rendezvous with and perform a variety of maneuvers in close proximity to the Multiple Paths Beyond Line of Sight Communications (MUBLCOM) satellite, without assistance from ground personnel. However, approximately 11 hours into what was supposed to be a 24-hour mission, DART detected that its propellant supply was depleted, and it began maneuvers for departure and retirement. Out of 27 defined mission objectives, DART met only 11 of those. NASA declared a “Type A” mishap, and convened a Mishap Investigation Board (MIB). The MIB found that DART had also collided with MUBLCOM 3 minutes and 49 seconds before its retirement. The MIB determined the underlying causes for this collision based on hardware testing, telemetry data analysis, and simulations. Even though no human casualties were reported, the mission failure resulted in a loss exceeding $1 million in government funds.

Software related causes played a significant role in this incident. The premature retirement of DART occurred due to a cycle of computational “resets” by the software throughout the mission, which triggered excessive thruster firings, and unexpectedly rapid fuel depletion. Also, incorrect velocity measurements from the primary GPS receiver were introduced into the software’s calculations during a reset, because the software fix for this known “bug” had never been implemented by the DART team. Furthermore, the design of the navigational software was inadequate, and the software failed to perform according to specification. One lesson learned from this accident is that any changes made to the software should be properly documented. Schedule pressure can result in inadequate testing of some late changes to the software. Also, any safety-critical software (such as the navigational software in this case) should not be overly-sensitive to erroneous data.

Loss of the Mars Polar Lander [9], [16]
The Mars Surveyor '98 program was comprised of two spacecraft launched separately: the Mars Climate Orbiter (formerly the Mars Surveyor '98 Orbiter), and the Mars Polar Lander (formerly the Mars Surveyor '98 Lander, and hereafter referred to as MPL). After an eleven month hyperbolic transfer cruise, MPL reached Mars on December 3, 1999. NASA lost contact with the spacecraft just prior to its scheduled atmospheric entry, and communication was never regained. The total cost was $120 million (not including the launch vehicle, and two DS2 microprobes) of which $110M was allocated to spacecraft development, and the other $10M to mission operations.

A special review board appointed by the Jet Propulsion Laboratory (JPL) Director identified that the most probable cause of the loss of MPL was due to a premature shutdown of the descent engines. Based on their findings, it is likely that one of the magnetic sensors attached to the landing legs tripped during descent, which falsely indicated that the spacecraft had touched down, resulting in a premature shutdown of the engines. The software, intended to ignore touchdown indications prior to the enabling of the touchdown sensing logic, was not properly implemented, and the spurious touchdown indication was retained. In addition, the touchdown sensing software was not tested with MPL in the flight configuration. As far as important lessons learned go, if a system is to be validated using simulation or other analyses, great care must be taken to ensure that the models underlying the analysis are suitably accurate and well-tested. Also, simulated stress testing and fault injection should be an integral part of the software testing process to discover hidden faults and establish the limits of the system.

Loss of the Mars Climate Orbiter [19]
A companion to the MPL (discussed above), the Mars Climate Orbiter, was also part of the Mars Surveyor' 98 program. The Mars Climate Orbiter was intended to enter an orbit at an altitude of 140–150 kilometers (460,000-500,000 feet) above Mars. However, a navigation error caused the spacecraft to reach as low as 57 kilometers (190,000 feet). The spacecraft was destroyed by atmospheric stresses and friction at this low altitude. The total cost incurred was $85 million (not including the launch vehicle) of which $80M was for spacecraft development and $5M was for mission operations.

The error in the navigation software was traced to a subcontractor which had used imperial units instead of the metric units specified by NASA. The computer controlling the spacecraft's thrusters had underestimated their force by a factor of 4.45. Although the software was adapted from an earlier Mars Climate Orbiter project, adequate testing was not performed prior to launch, and navigational data generated by the software was not cross-checked during flight. This event highlights the fact that even well-tested, proven software must still be thoroughly retested when deployed to a different environment.

Misplacement of a Satellite by Titan IV B-32/Centaur Launch Vehicle [12], [22]
On April 30, 1999, a Titan IV B configuration vehicle (Titan IV B-32) was launched with a mission to place a Milstar satellite in geosynchronous orbit. However, after the initial stages of flight, the vehicle began to unexpectedly roll. The Centaur stabilized itself during the subsequent coast phase, but 85% of the Reaction Control System (RCS) propellant had already been expended. During the second burn phase, the vehicle again began to roll, eventually losing pitch, and yaw control as well. Due to the premature depletion of RCS propellant, the attitude control system was unable to stabilize the vehicle. Because of the anomalies during the Centaur's flight, the Milstar satellite was placed in an unusable low elliptical orbit. The entire mission failed because of this misplacement, and the cost was about $1.23 billion.

The Accident Investigation Board concluded the root cause of the Titan IV B-32 mission mishap was due to the failure of the software development, testing, and quality/mission assurance process used to detect and correct a human error in the manual entry of a constant. An important lesson learned from this accident is that quality assurance engineers for safety-critical software must have a comprehensive understanding of the overall software development process to perform their work effectively. Also, any manually input critical data must be carefully verified using a rigorous process, and no single point of failure should be allowed.

Loss of Contact with the SOHO [21], [23]
The Solar and Heliospheric Observatory (SOHO) was deployed on December 2, 1995, and began normal operations in May, 1996 to study the Sun. However, flight controllers at NASA Goddard Space Flight Center (GSFC) lost contact with SOHO in the early morning of June 25, 1998. Control of SOHO was recovered subsequently; SOHO was re-oriented towards the Sun on September 16, and returned to its normal operations on September 25. However, even though control was eventually recovered, the one billion dollar spacecraft was at stake, and there was financial expense associated with recovery operations to establish contact with the lost spacecraft.

The incident was preceded by a routine calibration of the spacecraft's three roll gyroscopes (gyros). With respect to the software related causes, a modified command sequence in the onboard control software was missing a critical function to reactivate one of the gyros (Gyro A). The absence of this function caused the failure of an ESR (Emergency Sun Reacquisition) and set in motion the chain of events ending in the loss of telemetry from the spacecraft. Furthermore, another error in the software improperly left Gyro B in high gain mode, which generated the false readings that led to the inappropriate triggering of the ESR in the first place. Any modification to a ground operation procedure should be carefully evaluated to determine whether it will impact system reliability in either normal or contingency mode, and the applicable operational constraints for these modifications should be well-defined.

Explosion of Ariane 5 – Flight 501 [2]
Ariane 5 was an expendable launch system designed to deliver payloads into geostationary transfer orbit, or low Earth orbit. On June 4, 1996, the Ariane 5 launcher departed on its maiden flight. However, 37 seconds after its lift-off, at an altitude of about 3,500 meters, the launcher veered off its flight path, broke up, and exploded. The rocket had undergone a decade of development, and the expense is estimated at $7 billion. The rocket and cargo were valued at $500 million.

Shortly after lift-off, incorrect control signals sent to the Ariane 5 engines caused them to swivel. The resulting unsustainable stress on the rocket led to a breakup, and onboard systems automatically triggered a self-destruct. This malfunction can be directly traced to a software failure. The cause was a program segment that attempted conversion of a 64-bit floating point number to a 16-bit signed integer. The input value was larger than 32,767, and outside the range representable by a 16-bit signed integer, so the conversion failed due to an overflow. The error arose in the active and backup computers at the same time, resulting in the shutdown of both, and subsequent total loss of attitude control.

What this incident teaches us is that, for safety-critical software, in addition to testing what it should do, we should also test what it should not do. Instead of using the default system exception handler which simply shuts down the system, customized handlers should be implemented for various exceptions to ensure the safe operation of critical software systems. Whenever possible, software should be tested using real equipment rather than via simulations.

Emergency-Shutdown of the Hatch Nuclear Power Plant [5], [11]
The Edwin I. Hatch nuclear power plant was recently forced into an emergency shutdown for 48 hours after a software update was installed on a computer. The accident occurred on March 7, 2008 after an engineer installed a software update on a computer operating on the plant's business network. The software update was designed to synchronize data on both the business system computer, and the control system computer. According to a report filed with the Nuclear Regulatory Commission (NRC), when the updated computer rebooted, it reset the data on the control system, causing safety systems to errantly interpret the lack of data as a drop in water reservoirs that cool the plant's radioactive nuclear fuel rods. As a result, automated safety systems at the plant triggered a shutdown. Estimating a loss based on

electricity prices, the total cost to purchase electricity from external sources during the 48-hour shutdown period would be approximately $5 million. This does not include other operation-related costs.

The main cause of this accident was the installation of a software upgrade on a computer, without the knowledge of its impact on the entire system. System boundaries should be carefully defined, and the business network should be external to what might be safety critical. Although communication can be allowed between the business and control networks, it is better to accomplish this using a query mechanism rather than granting the former unrestricted access to the latter.

Miscalculated Radiation Doses at the National Oncology Institute in Panama [3], [6]
Panama's largest radiation therapy institution is the National Oncology Institute (Instituto Oncológico Nacional, ION). In March 2001, Panama's Ministry of Health asked the Pan American Health Organization (PAHO) to investigate some serious overreactions among cancer patients undergoing radiation therapy treatment at ION. A total of 56 patients were treated with improperly calculated doses. Only 11 were given doses within an acceptable margin of ± 5%. By August 2005, 23 of 28 at risk patients died. While it is unclear whether the patients would have died from cancer anyway, at least 18 were attributed to the radiation, mostly in the form of rectal complications.

The treatment planning system allowed a radiation therapist to draw on a computer screen the placement of metal shields (called "blocks"). Investigation revealed that the software only allowed the use of four shielding blocks, but the doctors at ION wished to use five. They thought they could trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors did not realize was that the software gave different answers in this configuration based on how the "hole" was drawn. Drawing it in one direction, the correct dose would be calculated; but drawing in another direction, the software recommended twice the necessary exposure. What this grave incident allows us to learn is that software (especially the safety-critical software) should be better designed such that input will only be accepted in a pre-specified format. Although it is a good idea to have flexible software, it has to be designed with care so that when users are not allowed to run the software based on their own assumptions; appropriate warning messages should be displayed.

Power-Outage across Northeastern U.S. and Southeastern Canada [8]
On August 14, 2003 at around 4pm EST, parts of the Northeastern United States and Southeastern Canada experienced widespread blackouts. The blackouts also resulted in the shutdown of nuclear power plants in the states of New York and Ohio, and air traffic was slowed as flights into affected airports were halted. More than 50 million people were affected, and the total cost was $13 billion. Regarding the causes of the incident, during some routine troubleshooting, a technician disabled a trigger that launched the state estimator every five minutes. Unfortunately, the technician forgot to turn the trigger back on after finishing with the maintenance.

While at first glance, this may seem unrelated to software; a robust system should however have a warning system, and send a notification when such a trigger is disabled. Also, a race condition in the system that set off the alarm caused it to lock up. Alarms that were to be sounded began to queue up with nothing to handle them. Because of this situation, the workers in the control room had no warning on the lines going down. Eventually, this incident crashed the server where it was hosted. An image of the locked up program and its queue of alarms were moved to a backup server, and an automated text was sent to a technician. Because the locked up program had only been moved over to a backup server, it did not take long for the same software fault to bring that server down too.

If a system fails due to a software fault, any identical system will also fail to the same fault. As a result, a standard form of "*redundancy*" (namely, having a backup system running exactly the same software) is not a solution for such cases.

Patriot Missile – Software Bug Led to System Failure at Dhahran, Saudi Arabia [20]
The Patriot is a surface-to-air defense missile system used by the United States Army. Using a series of complex, split-second computations, the computer calculates when to launch its missiles. In the case of Scuds, the system fires Patriots at each Scud. On the night of the 25th of February, 1991, a Patriot defense system operating at Dhahran, Saudi Arabia failed to track and intercept an incoming Scud. This Iraqi missile subsequently hit a U.S. Army barracks, killing 28 soldiers, and injuring another 98.

The Patriot battery at Dhahran failed to track and intercept the Scud missile because of a software problem in the system's weapons control computer. This problem led to an inaccurate tracking calculation that became worse the longer the system operated. At the time of the incident, the battery had been operating continuously for over 100 hours. By then, the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming Scud. An important lesson learned from this accident is that it is impossible to design a system that will work under every conceivable scenario. A system with well-defined limits of operation could fail, when it is operated outside of those limits. Also, just because an anomaly does not show up in thousands of hours of testing, it in no way implies that it may not show up in practice. Ironically, once identified, the bug was easy to fix, and the revised software was replaced in all 20 batteries in the war zone.

3. CONCLUSION

In this article, we have reviewed 15 catastrophic accidents, and have analyzed the roles that software has played in causing them. While the accidents may not have been caused by software alone, and may have been compounded by human error, undeniably software is either directly or indirectly responsible. We have also identified some of the useful lessons learned, and guidelines that must be followed to avoid the recurrence of such devastating accidents. This article in no way claims to be a complete listing of such accidents, but nonetheless provides a representative picture of how software that is faulty, or used in a faulty manner, might lead to not just loss in terms of money and time, but also the loss of life.

Is the solution to give up on our use of software for such purposes altogether Not likely. But certainly better testing procedures and practices need to be implemented, and this is especially true of software that is related to safety-critical systems. Also, software engineering and related curriculums at educational institutions should be updated to reflect the ever growing importance of software safety as a field.

REFERENCES

[1] Air France Flight 447 (http://www.airfrance447.com/about)
[2] Ariane 501 Inquiry Board, "Ariane 5, Flight 501 Failure," July 1996
[3] C. Borrás, "Overexposure of Radiation Therapy Patients in Panama: Problem Recognition and Follow-up Measures," *Rev Panam Salud Publica*, 20(2/3):173–187, 2006
[4] CNN, "TSA: Computer Glitch Led to Atlanta Airport Scare," April 21, 2006
[5] Environmental Protection agency, "Clear Skies in Georgia"
[6] Food and Drug Administration of U.S.A., "FDA Statement on Radiation Overexposures in Panama"
[7] L. Geppert, "Lost Radio Contact Leaves Pilots on Their Own," *IEEE spectrum*, 41(11):16-17, November 2004
[8] Great Northeast Power Blackout of 2003, http://www.globalsecurity.org/eye/blackout_2003.htm
[9] JPL Special Review Board, "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions," March 2000
[10] S. S. Krause, "Aircraft Safety: Accident Investigations, Analyses and Applications," second edition, McGraw-Hill, 2003
[11] B. Krebs, "Cyber Incident Blamed for Nuclear Power Plant Shutdown," Washington Post, June 5, 2008
[12] G. L. Lann, "Is Software Really the Weak Link In Dependable Computing?" The 41st IFIP WG 10.4 meeting, Saint John, US Virgin Islands, January 2002 (Workshop on *Challenges and Directions for Dependable Computing*)

[13] Los Angeles Times, “FAA to Probe Radio Failure,” September 17, 2004
[14] M. Nakao, “Crash of American Airlines Boeing”, December 1995
[15] National Transportation Safety Board, “Abstract on Korean Air Flight 801Conclusions, Probable Cause, and Safety Recommendations,” NTSB/AAR-99/02, August 1997
[16] NASA, Mars Polar Lander official website
[17] NASA, “Overview of the DART Mishap Investigation Results,” May 2006
[18] Official Guam Crash Site Information Web Center (http://ns.gov.gu/guam/indexmain.html)
[19] Report: Mars Climate Orbiter Mission Failure Investigation Board, http://marsprogram.jpl.nasa.gov/msp98/news/mco991110.html
[20] E. Schmitt, “Army is Blaming Patriot’s Computer for Failure to Stop Dhahran Scud,” New York Times, May 20, 1991
[21] SOHO Mission Interruption Joint NASA/ESA Investigation Board, “Final Report,” August 1998
[22] USAF Accident Investigation Board, “Titan IV B-32/Centaur/Milstar Report”
[23] K. A. Weiss, N. Leveson, K. Lundqvist, N. Farid, and M. Stringfellow, “An Analysis of Causation in Aerospace Accidents,” in *Proceedings of the 20th Digital Avionics Systems Conference*, Volume: 1, pp. 4A3/1-4A3/12, Daytona Beach, Florida, USA, October 2001